update Preisvergleich iPhone 3GS

Posted in **General** by admin on the September 8th, 2009

Nachdem die Sommereinführungsrunde vorbei war und ich mein Geld dann doch in einem Urlaub vollständige Verwendung fand, sitze ich nun ohne **iPhone 3GS** da. Hm, min. 35 € im Monat + 199 € (aka. 1.004 € über 2 Jahre) bei **T-Mobile** oder 39 € im Monat + 299 € (aka. 1.195 € über 2 Jahre) bei **Orange** statt meinen üblich 10-15 € zu zahlen, scheint mir doch etwas haarsträubend.

Doch diesen Montag die freudige Nachricht, 2**01 € Jubiläumsbonus** bei Orange, wow! Ok, klingt nach viel, aber was bringt es tatsächlich!? Das Geld wird einem – wie üblich – erst sukzessive gutgeschrieben, also 8,38- € pro Monat. Das klingt schon nach etwas weniger. Also ist es Zeit für ein kurzes Update – die neuen Preise für ein iPhone 3GS für die übliche 2 Jahresbindung sind in der folgenden Tabelle zusammengefasst.

Übersicht – Kosten des iPhone 3G S 16 GB in Österreich

Tarif	PhonePaket	Daten-paket	RabatteMin.		SMS	Daten-menge	Gesamt
Team 39	€ 69,-	€ 39,-	€ 14,-	€ 241,-	1000 + 300 EU	1100	3 GB	**€ 1.100,-**
Team 35	€ 119,-	€ 35,-	€ 14,-	€ 241,-	1000 + 200 EU	100	3 GB	**€ 1.054,-**
Team 30	€ 199,-	€ 30,-	€ 14,-	€ 241,-	1000 + 100 EU	100	3 GB	**€ 1.014,-**
Team 25	€ 199,-	€ 25,-	€ 14,-	€ 141,-	1000	100	3 GB	**€ 994,-**
Team 10	€ 399,-	€ 10,-	€ 14,-	€ 0,-	200	100	3 GB	**€ 935,-**
T-Mobile Tarife								
Call Austria	€ 249,-	€ 19,-	€ 5,-	€ 19,-	500 + 1.000 T-Mobile	0	100 MB	**€ 806,-**
	€ 199,-		€ 10,-				500 MB	**€ 876,-**
	€ 199,-		€ 15,-				3 GB	**€ 996,-**
Call Europe	€ 249,-	€ 29,-	€ 5,-	€ 29,-	1.000 (inkl. EU) + 1.000 T-Mobile	0	100 MB	**€ 1036,-**
	€ 199,-		€ 10,-				500 MB	**€ 1106,-**
	€ 199,-		€ 15,-				3 GB	**€ 1226,-**
Call & Surf Austria	€ 199,-	€ 39,-	0	€ 39,-	1.000 + 1.000 T-Mobile	1.000	1 GB	**€ 1096,-**
Call & Surf Europe	€ 149,-	€ 49,-	0	€ 49,-	2.000 (inkl. EU) + 1.000 T-Mobile	1.000	3 GB	**€ 1.276,-**
Classic	€ 199,-	€ 35,-	0	€ 35,-	1.000	0	3 GB	**€ 1.004,-**
Supreme	€ 149,-	€ 55,-	0	€ 55,-	1.000 intern, 1.000 Festnetz, 1.000 andere, 1.000 Box	1.000	3 GB	**€ 1.414,-**

Einige Änderungen der Gegebenheiten gegenüber Juni möchte ich hier noch anführen:

- den Tarif Team 15 gibt es nicht mehr
- alle Orangetarife haben weniger EU-Minuten
- das iPhone 3GS ist nicht mehr mir dem T-Mobile Tarif Call Easy kombinierbar
- bei Call Austria und Call Europe wird das iPhone nochmal um 50 € billiger, wenn man es mit einem Web&Walk Zusatzpaket kauft

0 Comments

iPhone 3G S – wo am billigsten in Österreich?

Posted in **Blogroll** by mip_digg on the June 19th, 2009

Ok, ich will es auch, das moderne geile ultracoole iPhone – und natürlich möchte ich ein 3G S. Schließlich muß es einfach das neueste sein. Hm, 16 GB sollten reichen, Provider ist mir eigentlich egal – und so wie viele Studenten bin ich natürlich bei Telering (oder Bob, A1, 3), auf jeden Fall nicht bei T-Mobile oder Orange.

Nach dem ich aber keinen Geldesel zu Hause habe, hier mal eine kurze Aufstellung, wo mein neues Lieblingshandy am günstigsten ist. Da die Tarife natürlich nicht sonderlich gut vergleichbar sind, habe ich Grundpreis phone, Paketpreis, Preis Datentarif, Nachlässe (z.B.: bis Ende des Jahres keine Grundgebühr), Online-Shop Rabatt, inkludierte Minuten, SMS und Datenmenge getrennt aufgeführt und schließlich den Gesamtpreis über 2 Jahre (die übliche Vertragsbindung) angegeben.

Übersicht – Kosten des iPhone 3G S 16 GB in Österreich

Tarif	PhonePaket	Daten-paket	Nach-lässe	Online-Rabatt	Min.	SMS	Daten-menge	Gesamt
Team Orange 39	€ 69,-	€ 19,50-	€ 14,-	0	€ 40,-	1000 + 400 EU	1100	3 GB	**€ 833,-**
Team Orange 35	€ 119,-	€ 17,50-	€ 14,-	0	€ 40,-	1000 + 300 EU	1100	3 GB	**€ 835,-**
Team Orange 30	€ 199,-	€ 15,-	€ 14,-	0	€ 40,-	1000 + 200 EU	1100	3 GB	**€ 855,-**
Team Orange 25	€ 199,-	€ 12,5-	€ 14,-	0	€ 40,-	1000 + 100 EU	100	3 GB	**€ 795,-**
Team Orange 15	€ 399,-	€ 7,5-	€ 14,-	0	€ 40,-	300 min. oder SMS	100	3 GB	**€ 875,-**
	über den inkl. Mengen kann man immer umsonst mit Orange telefonieren & SMSen								
Call & Surf Austria	€ 149,-	€ 39,-	0	0	€ 39,-	1000 + 1000 T-Mobile	1000	1 GB	**€ 1046,-**
Call & Surf Europe	€ 49,-	€ 49,-	0	0	€ 49,-	2000 (AT+EU) + 1000 T-Mobile	1000	3 GB	**€ 1176,-**
Classic	€ 149,-	€ 35,-	0	0	€ 35,-	1000	1000 (unter 27)	3 GB	**€ 954,-**
Supreme	€ 49,-	€ 55,-	0	0	€ 55,-	1000 T-Mobile, 1000 Festnetz, 1000 andere, 1000 Box	1000	3 GB	**€ 1314,-**
Call Easy	€ 479,-	€ 9,-	€ 5,-	€ 54,-	0	100 + 1000 T-Mobile	0	100 MB	**€ 761,-**
Call Easy			€ 10,-					500 MB	**€ 881,-**
Call Easy			€ 15,-					3 GB	**€ 1001,-**
Call Austria	€ 249,-	€ 19,-	€ 5,-	€ 114,-	€ 19,-	500 + 1000 T-Mobile	0	100 MB	**€ 692,-**
Call Austria			€ 10,-					500 MB	**€ 812,-**
Call Austria			€ 15,-					3 GB	**€ 932,-**
Call Europe	€ 249,-	€ 29,-	€ 5,-	€ 174,-	€ 29,-	1000 (inkl. EU) + 1000 T-Mobile	0	100 MB	**€ 862,-**
Call Europe			€ 10,-					500 MB	**€ 982,-**
Call Europe			€ 10,-					3 GB	**€ 1102,-**

Also wenn man keine SMS braucht und nur wenig Datenvolumen ist das iPhone bei T-Mobile am günstigsten, wenn man ein bißchen was machen will, dann eigentlich bei Orange. Zu bedenken ist, dass das bei Orange nur aufgrund der Halbpreis-Aktion geht – und die läuft (offiziell) nur bis 30.6. Die Nachlässe bei T-Mobile sind auch für einen Abschluß vor dem 30.6 gerechnet, also man spart sich 6 mal die Grundgebühr bis Jahresende.

Latest Digg Stories

Anscheinend sind SMS und Datenmengen bei T-Mobile besonders teuer, auch ist das iPhone über die 2 Jahre Bindefrist mit einem Durchschnittspreis von € 962,7- um einiges teurer als bei Orange (€ 838,6-).

So, jetzt hoffe ich nur, dass ich richtig gerechnet hab und dass ihr brav auf die Werbung klickt, dann kann ich mir vielleicht auch mal ein iPhone leisten  .

1 Comment

Xtirer 0.2 Introduction Summary

Posted in **Xtirer** by admin on the July 15th, 2008

Ok, Xtirer 0.2 is out and i've already issued some posts showing a [redacted] path to usage. And here is just are summary with links to all parts of the introduction:

- **Installing Xtirer (Unpack, upload, configure)**
- **Simple Reading from a MYSQL Database and how the XML-Output is generated**
- **Advanced Reading introduction (distinct, order, limit, offset, etc.)**
- **Finetuning your database reading results with the selector object**
- **Storing data into your MYSQL Database (insert, update, delete)**
- **And finally some helpers**

You'll find Xtirer at **http://sourceforge.net/projects/xtirer**
In case of any questions lieve a comment or post on the xtirer forum at sourceforge.

1 Comment

Xtirer helper tags: condition, sql & dirreader

Posted in **Xtirer** by admin on the July 15th, 2008

My introductory to Xtirer 0.2 is almost finished – so this time it's all [redacted] about the helper objects, that went handy during my latest development projects. Check out the posts about reading data from MYSQL via Xtirer: **Intro**, **part 2** & **part 3** and the guide **how to write data to the database**.

Let's start with the `sql` object:

```
<sql
 function="SQL_FUNCTION_NAME"
 param="SQL_FUNCTION_PARAMETERS">
```

This helper is quite straight forward in terms of setup and it's effect on the MYSQL query. Take a look at the example below:

```
<?xml version="1.0" encoding="UTF-8"?>
<test>
 <xml_object name="simple_sql">
    <sql param="1+1"/>
  </xml_object>
  <xml_object name="other_simple_sql">
    <sql function="CONCAT" param="'hello','world'"/>
  </xml_object>
  <xml_object name="more_sql" source="TABLE.animals">
    <xml_object name="owner" valueof="COLUMN.owner">
      <sql function="CONCAT" param="'Owner:',#this"/>
    </xml_object>
    <xml_attribute name="namelength" valueof="COLUMN.owner">
      <sql function="LENGTH" param="#this"/>
    </xml_attribute>
```

```
  </xml_object>
</test>
```

SQL would be:

```
SELECT (1+1)

SELECT CONCAT('hello','world')

SELECT CONCAT('Owner:',owner), LENGTH(owner) FROM animals
```

Ok, i think you get the idea. Function defines the SQL Function to use (**see the MYSQL function section**).
`Param` defines the parameter list. Please use single quotes to enclose strings. You may use “#this” for including the parent’s sql reference.

Next helper: condition

```
<condition
 variable="XTIRER_EXPRESSION"
 value="XTIRER_EXPRESSION"
 compare="eq|neq|gr|sm|gre|sme">
```

Some sample code:

```
<?xml version="1.0" encoding="UTF-8"?>
<test>
  <xml_object name="save" value="2" type="integer"/>
  <condition variable="$parent.save" value="1" compare="gre">
    <message>success!!</message>
  </condition>
  <condition variable="$parent.save" value="0">
    <message>error!</message>
  </condition>
</test>
```

The `condition` object enables conditional output, which is especially helpful when writing data to the database. The `db_value` object always returns the insert id (except of delete actions), so checking for successful inserts is quite easy.

`Variable` and `value`-path again can be Xtirer expressions; Starting with # or $ for the SQL Name or the nodes value you can reference objects throughout the xml-tree. But please be forgiving – this is one of the beta things.

Note: If the `condition` object contains some SQL action (aka. `xml_object`, `xml_attribute`, `db_value`, etc.) these commands are executed regardless the result of the condition, only the output is filtered!

Last tag for this post: `dirreader`

```
<dir_reader
 name="SOME_NAME"
 path="DIRECTORY_PATH"
 http="true|false"
 subdir="true|false"
 limit="NUMBER"
 offset="NUMBER"
 order="ascend|descend|alphabetuc"
 output="[name|type|size|owner|group|date|dir|full|number]"
 excludeon="name|type|size|owner|group|date|dir|full|number">
```

As the name already suggests it, this has nothing to do with SQL, but in a lot of web projects you sometimes need directory listings. So this object gives you the listing in XML formatted ways. Consider the following prototype files:

```
<?xml version="1.0" encoding="utf-8"?>
<list>
```

```
  <dir_reader name="file" path="dir" http="true" order="numeric" ou
</list>
```

Output for dir being some path:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<list>
 <file type="DS_Store" number="1">
  <name><![CDATA[.DS_Store]]></name>
</file> <file type="php" number="2">   <name><![CDATA[array_splice_
 <file type="php" number="4">
 <name><![CDATA[config.example.inc.php]]></name>
</file>
<file type="php" number="5">
<name><![CDATA[config.inc.php]]></name>
</file>
<file type="html" number="6">
<name><![CDATA[debugfooter.html]]></name>
 </file>
</list>
```

The function of most of the parameters is quite obvious, so let's discuss the two main attributes you might need: output and excludeon. The output attribute takes one or more values delimited by coma and control the output per file found. With excludeon you might define one value on which Xtirer should filter the files.

Let's consider the following: You have a table in your database containing all files of a gallery with their path and you want to provide a file upload to add additional pics. The user selects the uploaded and adds it to the database. So you would need a list of files which are in the upload-directory but are not present in the database. So you would use something like this:

```
<?xml version="1.0" encoding="utf-8"?>
<list>
  <dir_reader name="file" path="dir" http="true" order="numeric" ex
    <xml_object name="exclude" source="TABLE.gallery" valueof="COLU
  </dir_reader>
</list>
```

Xtirer reads the file list from the directory specified by *dir* and excludes all files that are stored in the table gallery via comparing the full file path to the entries in the table. The child node has to be named "exclude"!

I think there is enough to explore in Xtirer and i hope you find something useful for ya. Every comment and feedback welcome!

1 Comment

store data to MYSQL with Xtirer

Posted in **Xtirer** by admin on the July 10th, 2008

In my **last post**, i've shown how to finetune data reading by using Xtirer's `selector` object. So you should already be able to get your values from the **MYSQL database**. Today, it's all about writing and deleting data to the tables. Let's assume we like to store data to a table containing the team member's data. So your Xtirer prototype file could look somewhat like this:

```
<?xml version="1.0" encoding="utf-8"?>
 1: <datastore>
 2: <db_value field="TABLE.teamlist" name="save" type="integer" act
```

```
 3:  <db_value field="COLUMN.id" http="personid" type="integer"/>
 4:  <db_value field="COLUMN.name" http="name"/>
 5:  <db_value field="COLUMN.position" http="position"/>
 6:  <db_value field="COLUMN.email" http="email"/>
 7:  <db_value field="COLUMN.description" http="description"/>
 8:  <db_value field="COLUMN.image" http="img_id" type="integer"/>
 9: </db_value>
10: </datastore>
```

And here goes the definition of the `db_value` object:

```
<db_value
 name="SOME_NAME"
 field="XTIRER_TABLE_REFERENCE|XTIRER_COLUMN_REFERENCE"
 action="REPLACE|UPDATE|DELETE"
 type="number|string"
 http="HTTP_VARIABLE_NAME"
 value="SOME_VALUE">
```

I think it is quite obvious what is happening there:
Line 1: the datastore tag is just a wrapper – so no functionality
Line 2: the first db_value object – the parent- should define the table to write to
Line 3-8: database columns to write to and http variable names to get the data from.

Things to note:
the type attribute triggers quoting or not for the value: 1 vs. ‘1′
adjust the desired write action by changing the action attribute of the parent `db_value` object
there is no “**INSERT**” command ’cause “**REPLACE**” does the same for new entries, but can be used to change data of already existing rows when a unique key you like to write already exists. That’s why you usually don’t need a `selector` object.

Ok, but what’s the notion behind the name- and type-attribute of the parent `db_value` object in line 2!?
Of course in most cases you should be interested of the result of the writing action. So the `db_value` returns the **LAST_INSERT_ID**. Hm, and there should be some way to exploit this information.

So, next code shows how my writing prototype files usually look like:

```
<?xml version="1.0" encoding="utf-8"?>
 <datastore>
 <db_value field="TABLE.teamlist" name="save" type="integer" action
  <db_value field="COLUMN.id" http="personid" type="integer"/>
  <db_value field="COLUMN.name" http="name"/>
  <db_value field="COLUMN.position" http="position"/>
  <db_value field="COLUMN.email" http="email"/>
  <db_value field="COLUMN.description" http="description"/>
  <db_value field="COLUMN.image" http="img_id"/>
 </db_value>
 <condition variable="$parent.save" value="1" method="gre">
  <message>success!!</message>
 </condition>
 <condition variable="$parent.save" value="0">
  <message>error!</message>
 </condition>
</datastore>
```

I’d like to finish for this post, but i promise to handle the helper objects such as the `condition`, `sql` or `dirreader` object in the next. So, check back next time! Have fun.

0 Comments

data reading & Xtirer – part 3: Select

Posted in **Xtirer** by admin on the July 8th, 2008

Weekend is over, so let's get it on; After explaining the `xml_object` | `xml_attribute` in detail in **part 2** of the Xtirer 0.2 Introduction, it is time to get hands on some fine tuning of row selection.

Obviously, the `WHERE` clause is missing, so to put it simple: with the `selector` tag you can define the value of columns for the rows to be selected.

Definition:

```
1: <selector
2:  variable="XTIRER_COLUMN_REFERENCE"
3:  value|array="STATIC_VALUE"
4:  valueof|arrayof="XTIRER_COLUMN_REFERENCE"
5:  http="false|true"
6:  compare="eq|neq|gr|sm|gre|sme"
7:  type="number|string"
8:  boolean="AND|OR|NOR|XOR|NAND">
```

- `variable`: Column name which should be set/defined.
- `value|array`: static value or comma seperated array.
- `valueof|arrayof`: Column name of a parent's query result, which to use for this selection
- `http`: if true, valueof/arrayof is interpreted as a http variable name and that content is used
- `compare`: Comparison method between variable and value
- `type`: string values (default) are always quoted; numbers not.
- `boolean`: if multiple selectors are used they can be combined with different logical operators; AND is default.

The unique feature of the `selector` object is that it does not inherit the data source from the parent. It roots itself in the grandparent. But let's see some examples for explanation:

```
<container>
  <xml_object name="animal" source="TABLE.animals">
    <selector variable="COLUMN.id" value="httpid" http="true" type=

    <xml_object name="name" valueof="COLUMN.name"/>
    <xml_attribute name="birth" valueof="COLUMN.birth"/>
  </xml_object>
</container>
```

What will happen? Yep, you're right:

Xtirer tries to read a variable called *httpid* from the http variables and will build the following query:

```
SELECT name,birth FROM animals WHERE id = [httpid];
```

Output and the rest should be pretty straight forward.
If you use `arrayof` instead of `valueof` Xtirer assumes you use a comma-seperated array and the above query would look like:

```
SELECT name,birth FROM animals WHERE id IN ([httpid])
```

Ok, next – this:

```
<container>
  <xml_object name="animal" source="TABLE.animals">
    <xml_object name="name" valueof="COLUMN.name"/>
    <xml_attribute name="birth" valueof="COLUMN.birth"/>

    <xml_object name="funky" source="TABLE.funkystuff">
```

```
      <selector variable="COLUMN.id" valueof="COLUMN.id" type="numb
      <xml_attribute name="num" valueof="COLUMN.number"/>
    </xml_object>
  </xml_object>
</container>
```

Here it gets obvious, that the `selector` object has to inherit from the grandparent; Otherwise the Where-Clause of the secound SQL-Query would make no sense.
First, Xtirer will execute the standard query:

```
SELECT name, birth, id FROM animals
```

And then for every result row the following gets read from the database:

```
SELECT num FROM funkystuff WHERE id <=> [id from animals]
```

The Xtirer_COLUMN_Reference in the `valueof` attribute is adressing the animals-table, whereas the `variable` attribute references the id column of the funkystuff-table.

If you use `arrayof` instead of `valueof` the result from the database is treaded as an array. This one advantage of Xtirer above a **standard SQL JOIN**.

To make it a bit more confusing, but shorter you could simply add the funkystuff-number to the animal object by replacing the

```
<xml_object name="funky"...
```

with:

```
<xml_attribute name="funky" source="TABLE.funkystuff" valueof="COLU
  <selector variable="COLUMN.id" valueof="COLUMN.id" type="number"/
</xml_attribute>
```

To not be confused you could also write:

```
<selector variable="COLUMN.id" valueof="TABLE.animals.COLUMN.id" ty
```

but you don't have to.

I hope, my explanations are intelligible to all and you still like Xtirer.

Next, i'll gonna turn to data writing and deleting. So check back soon, cheers!

2 Comments

reading data with Xtirer – part 2

Posted in **Xtirer** by admin on the July 4th, 2008

Ok, last time i've shown the first really basic functions of Xtirer reading data from MYSQL and generating an XML output. So this post is going more into detail showing more possibilities.

First of all here the definition of the xml-object / xml-attribute. Both objects are defined equally, besides the output being different.

```
<xml_object|xml_attribute
 name="outputname"
 [[distinct="true|false"
 source="Xtirer_table_reference"
 value="STATIC_VALUE"
 valueof="XTIRER_COLUMN_REFERENCE"
```

```
limit="MAX_OF_ROWS"
offset="ROW_OFFSET"
order="ASC|DESC|ASCENDING|DESCENDING"]]>
```

You see every attribute except the name attribute is optional. Now let's explain each attribute and its effect in terms of SQL-Syntax and XML Output. The relevant part is highlighted in red.

- **name**: defines the name of the output object – tag or attribute name

```
Xtirer: <xml_object name="myxml"/>
SQL: - (nothing)
Output: <myxml/>
```

- **value**: defines a static value to be the content of the xml-object or value of the xml-attribute

```
Xtirer: <xml_object name="myxml" value="somevalue"/>
        <xml_attribute name="myvalue" value="someother"/>
SQL: - (nothing)
Output: <myxml><![CDATA[somevalue]]></myxml>
        <... myvalue="someother" ..
```

- **valueof**: (* assuming some parent defined the TABLE being "animals") defines the database table column from where the content | value of the object | attribute shall come.

```
Xtirer: <xml_object name="myxml" valueof="COLUMN.owner"/>
        <xml_attribute name="myattr" valueof="COLUMN.owner"/>
SQL: SELECT owner FROM animals;
Output: <myxml><![CDATA[database value of column
owner]]></myxml>
        <... myattr="database value" ..
```

- **source**: defines the database table where the data shall come from. Objects inside inherit the source, so pls. only define once.

```
Xtirer: <xml_object name="myxml" source="TABLE.animals"
valueof="COLUMN.birth"/>
SQL: SELECT birth FROM animals;
Output: <myxml><![CDATA[database value]]></myxml>
```

- **distinct**: if only distinct values of one column are required set distinct true.

```
Xtirer: <xml_object name="myxml" valueof="COLUMN.owner"
distinct="true"/>
SQL: SELECT distinct owner FROM animals;
Output: <myxml><![CDATA[database value]]></myxml>
```

- **order**: if results shall be ordered ascending or descending indicate the ordering column with "order" and the direction. Possible values are ASC,DESC,ASCENDING,DESCENDING in upper and lower cases.

```
Xtirer: <xml_object name="myxml" valueof="COLUMN.owner"
order="ASC"/>
SQL: SELECT owner FROM animals ORDER BY owner ASC;
Output: <myxml><![CDATA[database value]]></myxml>
```

- **limit**: if you like to limit the number of result objects use the limit attribute

```
Xtirer: <xml_object name="myxml" valueof="COLUMN.owner"
LIMIT="5"/>
SQL: SELECT owner FROM animals LIMIT 5;
Output: <myxml><![CDATA[database value]]></myxml>
```

- **offset**: if you would like to skip some of the results, e.g. get the 11-20 position of the charts use the offset value. If no limit is defined the offset is ignored.

```
Xtirer: <xml_object name="musictitle"
value="COLUMN.musictitle" LIMIT="10" OFFSET="10"/>
SQL: SELECT musictitle FROM charts LIMIT 10,10;
```

```
Output: <musictitle><![CDATA[database value]]></musictitle>
```

lol, getting a bit cody here, but don't be scared away. All these possibilities are really handy and the usage is very straight forward.

Ok, the final thing to understand the Xtirer **xml_object** | **xml_attribute** tags is "source aka sql nesting". Let's have a look at the next Xtirer Prototype file:

```
 1: <animallist>
 2: <xml_object name="pat" source="TABLE.animals">
 3:   <xml_object name="name" valueof="COLUMN.name"/>
 4:   <xml_attribute name="species" valueof="COLUMN.species"/>
 5:
 6:   <xml_object name="funky" source="TABLE.funkystuff">
 7:     <xml_attribute name="number" valueof="COLUMN.number"/>
 8:   </xml_object>
 9:
10:   <xml_attribute name="birth" valueof="COLUMN.birth"/>
11:  </xml_object>
12: </animallist>
```

This is a reduced version of the animals example in the introduction to Xtirer database reading. I've just added an xml_object named funky and its attribute.
What will happen?

Xtirer will scan through the XML-Prototype file and do the following:
1. Make a basic xml tag named "animallist"
2. Make an SQL Query "SELECT name,species,birth FROM animals"
3. Replicate the xml_object as many times as there are resulting rows in the DB.
4. For every result row execute the SQL Query "SELECT number FROM funkystuff"
5. Replicate the xml_object funky for every query and every result row

So the output will be:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<animallist>
<pat species="dog" birth="2008-07-11">
<name><![CDATA[chaco]]></name>
<funky number="1"/>
<funky number="2"/>
<funky number="34"/>
<funky number="500"/>
</pat>
<pat species="cat" birth="2007-07-20">
<name><![CDATA[lurch]]></name>
<funky number="1"/>
<funky number="2"/>
<funky number="34"/>
<funky number="500"/>
</pat>
<pat species="ape" birth="1997-07-04">
<name><![CDATA[chimpike]]></name>
<funky number="1"/>
<funky number="2"/>
<funky number="34"/>
<funky number="500"/>
</pat>
</animallist>
```

Things to note:
1. all objects in line 3,4 and 10 inherit the source defined in line 2.
2. xml-attribute in line 7 takes source from line 6
3. Query defined with tags of line 6+7 are executed as often as results are found for the other SQL query

Ok, i stop here for now. Next time i'll introduce the selector tag to select specific rows in a table.

Feel free to send me your feedback on Xtirer and check back soon!

1 Comment

Next Page »